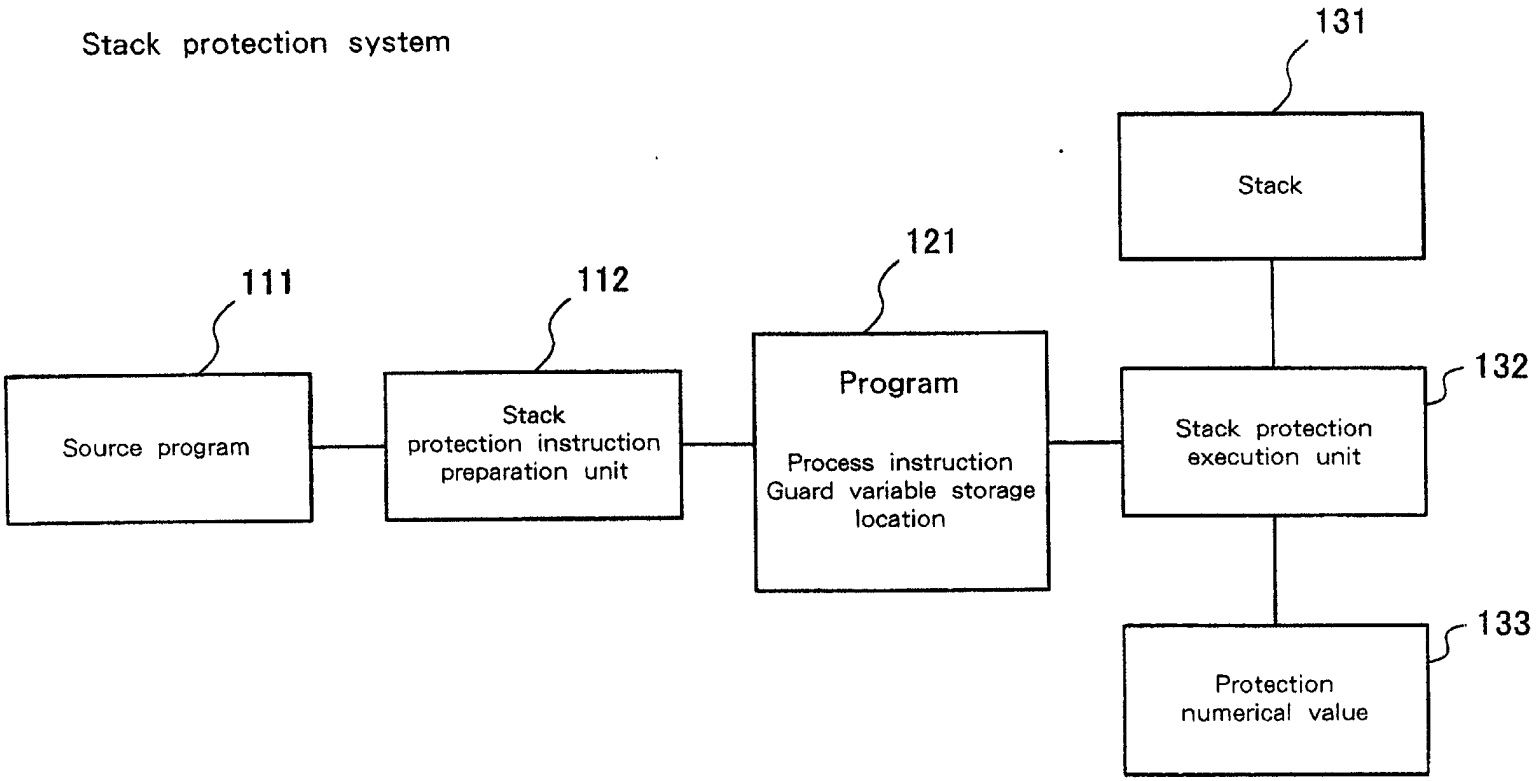
Stack protection system
131
Stack
121
111
112
Program
Process instruction
Guard variable storage location
132
Source program
Stack protection instruction preparation unit
Stack protection execution unit
133
Protection numerical value

Fig. 1

Memory pattern
when guard variable is stored

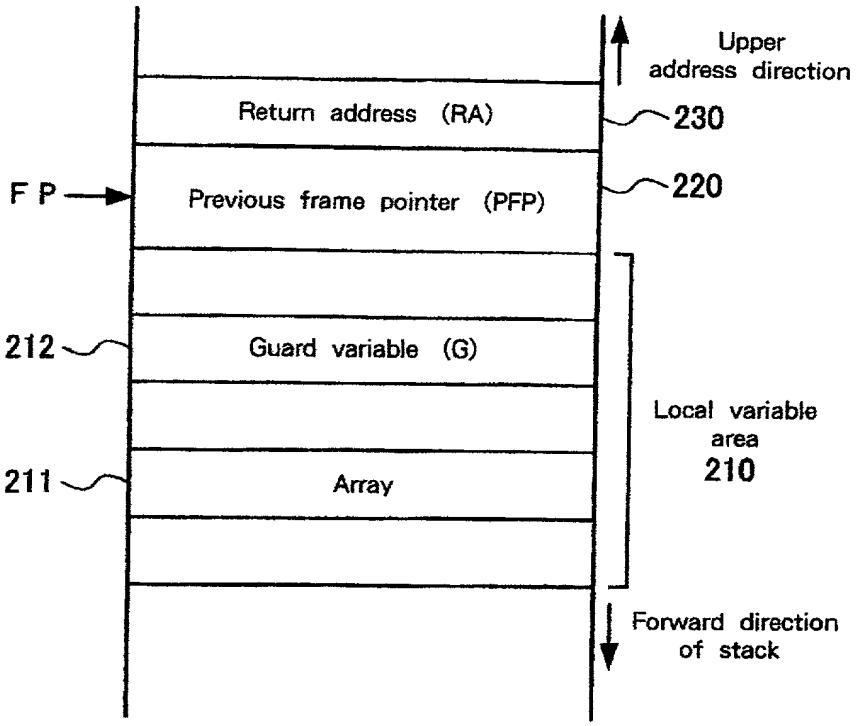

Fig. 2

Processing of stack protection instruction preparation unit 112

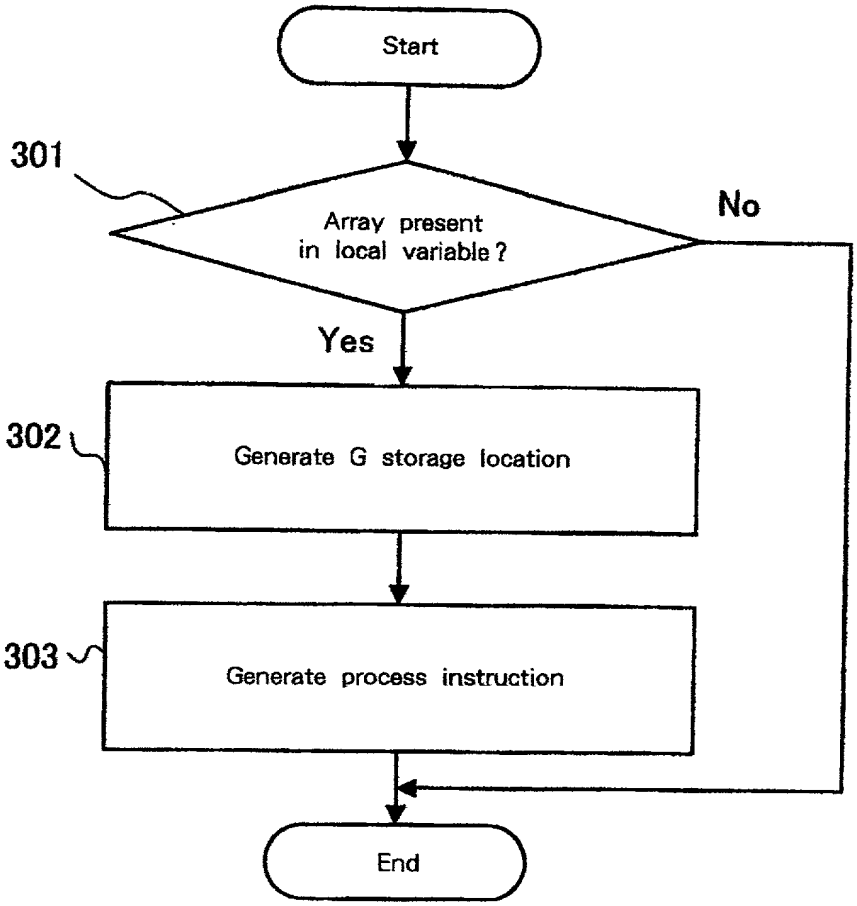

Fig. 3

Processing of stack protection execution unit 132

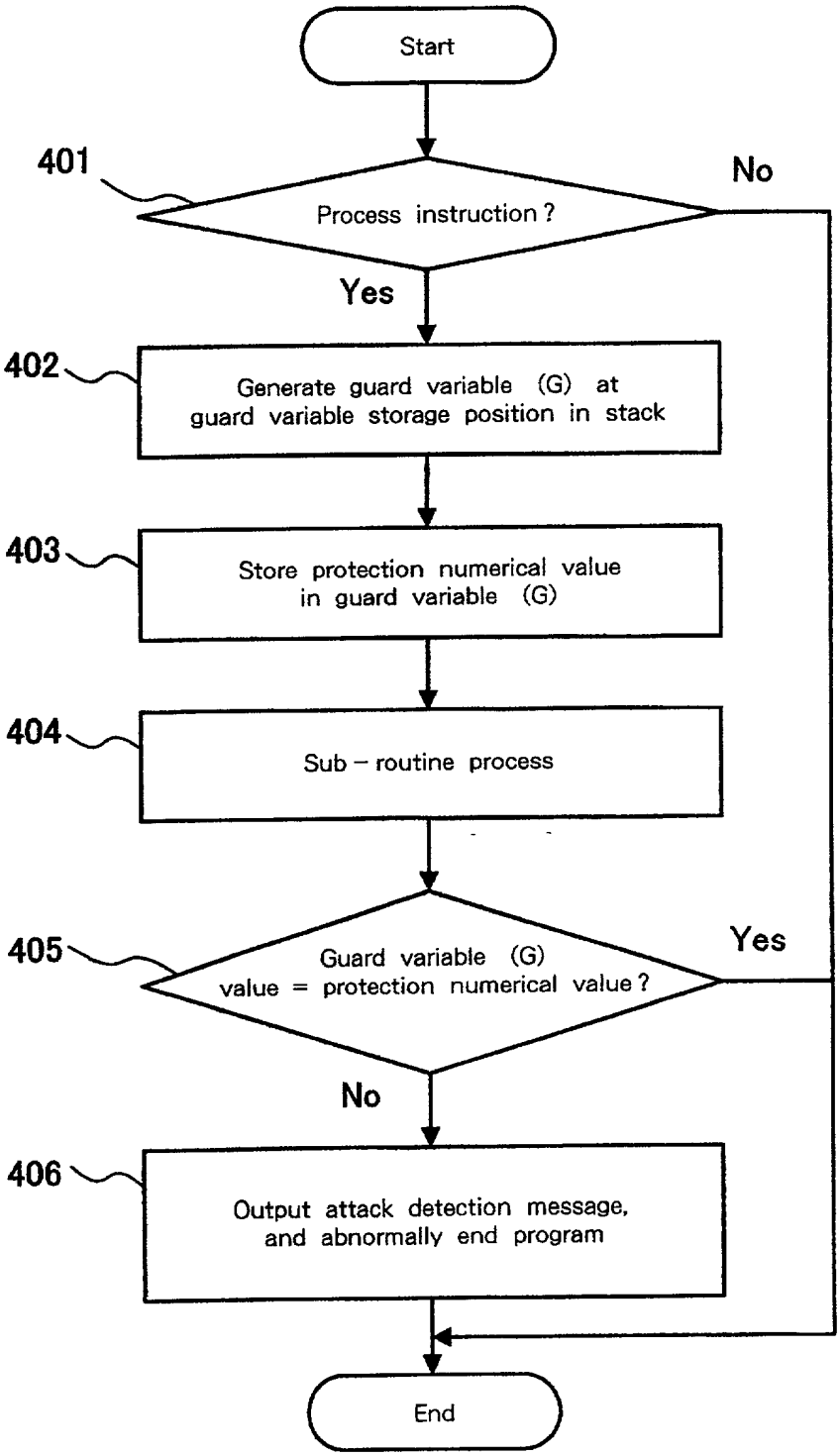

Fig. 4

- Variable declaration

```
volatile int   guard;
```

- Function entrance

```
gv = guard_value;
```

- Function exit

```
if (gv!= guard_value){
    /*output error log */
    /*halt execution */
}
```

Fig. 5

```
void foo()
{
        volatile int guard;      ←— 601
        char buf[128];

        gv = guard_value;        ←— 602
        ---.

        strcpy (buf, getenv ("HOME"));
        ---.

        if (gv!= guard_value){
            /*output error log */     ←— 603
            /*halt execution */
        }
}
```

Fig. 6

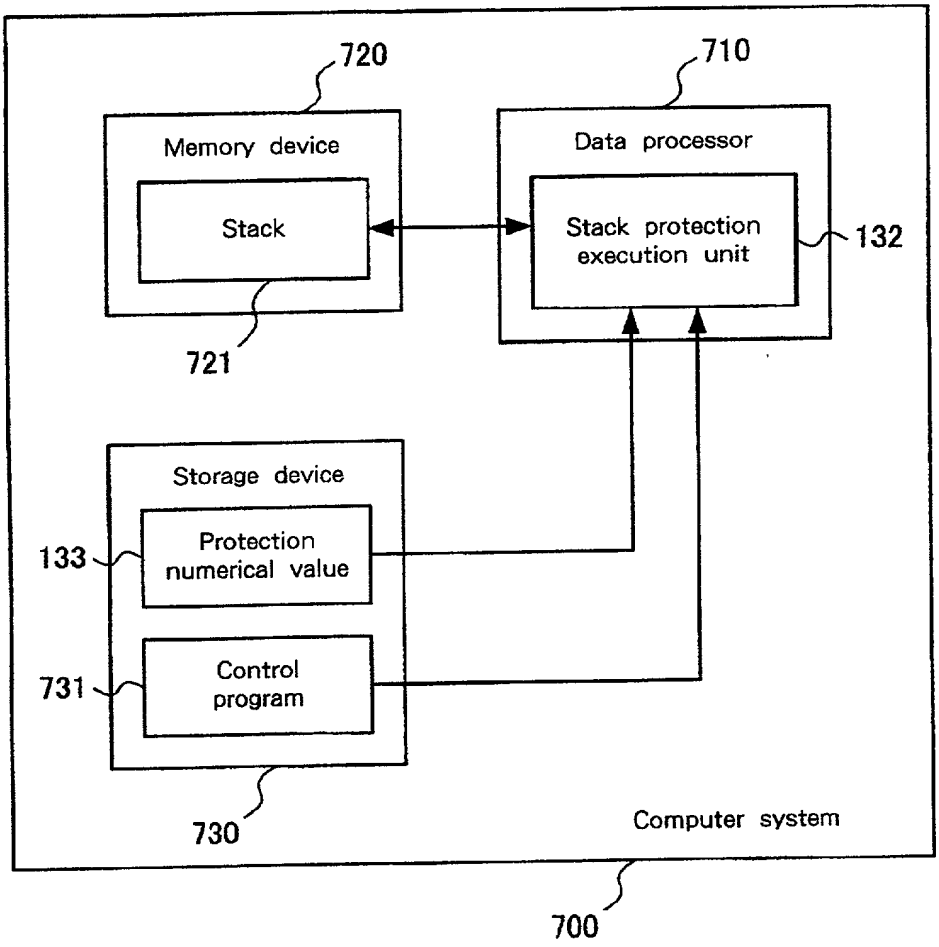
720
710
Memory device
Data processor
Stack
Stack protection execution unit
132
721
Storage device
133
Protection numerical value
731
Control program
730
Computer system
700

Fig. 7

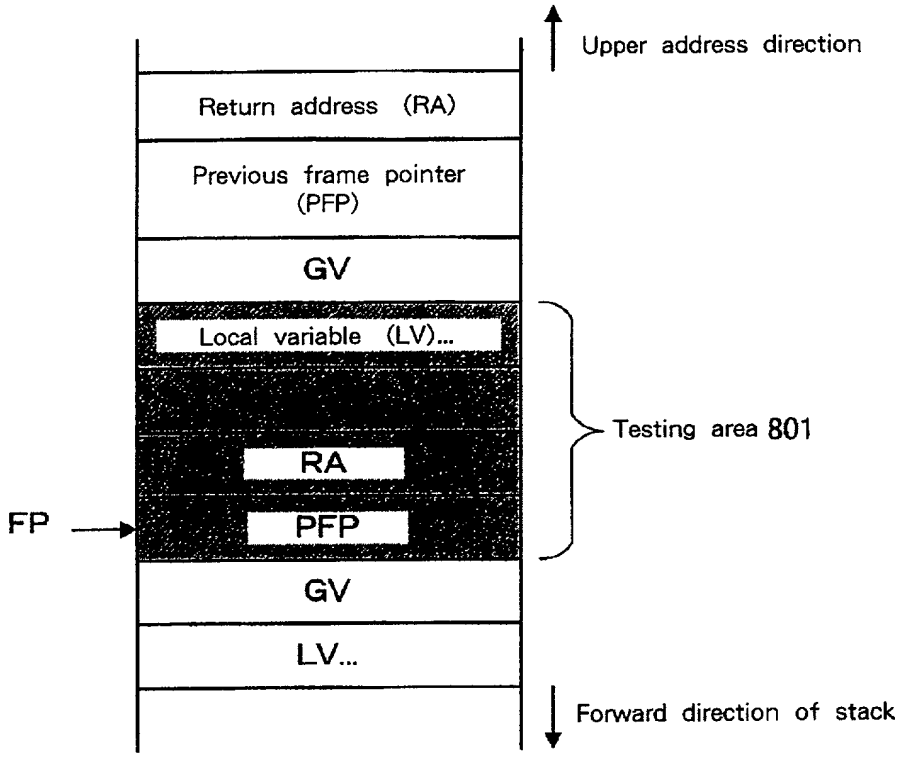
Upper address direction
Return address (RA)
Previous frame pointer (PFP)
GV
Local variable (LV)...
RA
PFP
FP
Testing area 801
GV
LV...
Forward direction of stack

Fig. 8

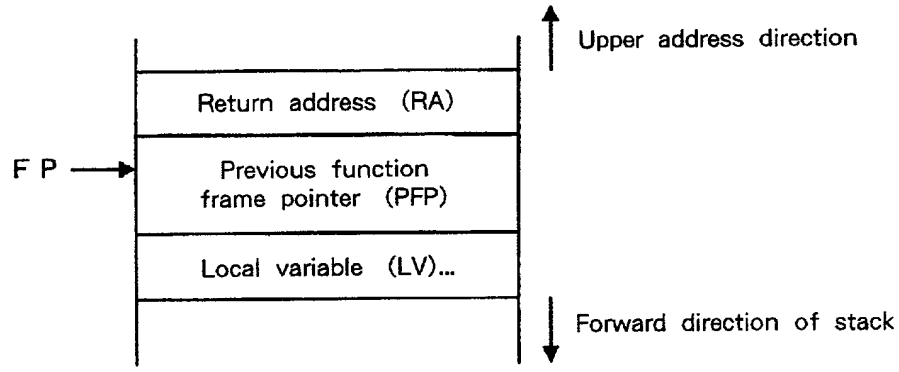
Upper address direction
Return address (RA)
F P
Previous function frame pointer (PFP)
Local variable (LV)...
Forward direction of stack

Fig. 9

```
void foo()
{
        char buf[128];
        ---
        strcpy (buf, getenv ("HOME"));
        ---
}
```

Fig. 10

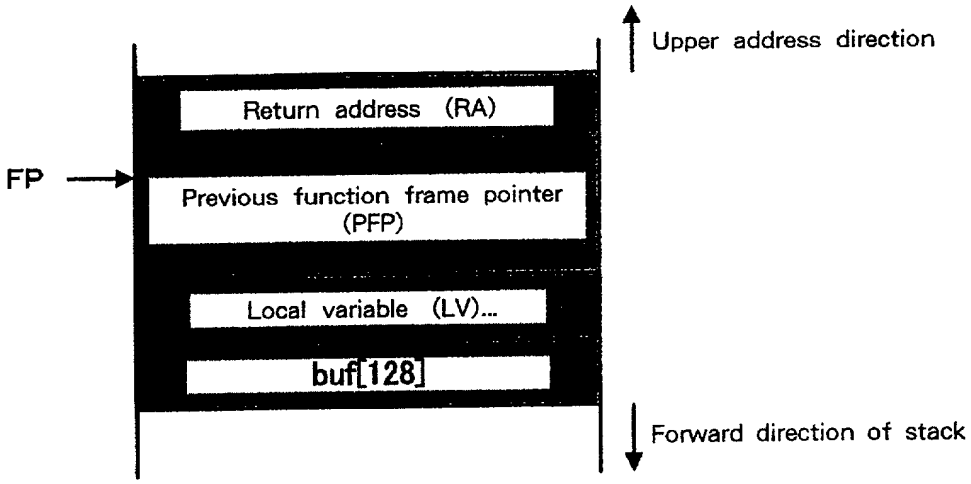

Fig. 11